МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Изучение деревьев

Студент гр. 8304 _____ Мешков М.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

Цель работы.

Получить опыт работы с деревьями, изучить на практике особенности их реализации.

Постановка задачи.

Вариант 5-д.

Задано бинарное дерево b с произвольным типом элементов. Используя очередь и операции над ней, напечатать все элементы дерева b по уровням: сначала из корня дерева, затем (слева направо) из узлов, сыновних по отношению к корню, затем (также слева направо) из узлов, сыновних по отношению к этим узлам, и т. д.

Описание алгоритма.

Для решения поставленной задачи для начала нужно считать данные. В начале программа запрашивает у пользователя бинарное дерево в виде скобочного представления. Затем осуществляется преобразование введенной строки в дерево с помощью специально написанной функции. Затем осуществляется вывод этого дерева по уровням, это выполняется с помощью очереди - в начале в очередь помещается корень дерева, затем запускается цикл, который берет из очереди элемент, заносит его значение в строку результата и помещает его детей в очередь, этот цикл продолжается до тех пор, пока очередь не станет пуста.

Описание основных структур данных и функций.

Функция main осуществляет всё взаимодействие с пользователем, она приводит к созданию всех остальных используемых структур данных.

Специально для решения данной задачи был написан класс Queue, реализующий очередь с использованием динамической памяти.

Этот класс имеет следующий интерфейс:

- push - помещает новый элемент в конец очереди,

- first - возвращает первый элемент очереди,
- pop - убирает из очереди первый элемент,
- isEmpty - позволяет быстро узнать пуста ли очередь.

Также специально для решения данной задачи был написан класс BinTree, реализующий бинарное дерево с использованием динамической памяти.

Этот класс имеет следующий интерфейс:

- rootValue - возвращает значение корня дерева,
- setRootValue - устанавливает значение корня дерева,
- left - возвращает левое поддерево.
- right - возвращает правое поддерево.

Для преобразования скобочного представления бинарного дерева в экземпляр класса BinTree была написана функция loadBinTree, которая возвращает перечисление BinTreeLoadingStatus, которое позволяет узнать, было ли успешно преобразовано дерево, и если нет, то узнать причину.

Для получения строки с элементами дерева, записанными по уровням, была написана функция binTreeByLayers.

Тестирование.

Программа была успешно протестирована. Ниже приведены основные проверочные входные данные.

Ввод	Вывод
(a (b(c))(d))	a b d c
(a(b (c#(d)))(e(f (g)(h))(i)))	a b e c f i d g h
(1 (2(4) (5))(3(6)))	1 2 3 4 5 6
(aa (j8(b (di))(c(e)(fk)))(g #(h#(i))))	aa j8 g b c h di e fk i
(a# #)	a
(a (b(d)(e#(h)))(c(f (i)(j(m)(n(q)(r))))(g (k#(o))(l(p)))))	a b c d e f g h i j k l m n o p q r
(a###)	Error: expected the closing bracket at the position 5.
(a(b))))))	Warning: extra characters after the tree expression. a b

)(	Error: expected the opening bracket at the position 1.
(a(b	Error: unexpected the end of the tree expression.

Выводы.

В ходе выполнения работы был получен опыт работы с бинарным деревом в контексте выполнения данной задачи, изучены методы работы с ним. Была написана реализация бинарного дерева через динамическую память.

ПРИЛОЖЕНИЕ А

Файл main.cpp

```
#include <iostream>
#include <memory>
#include <optional>
#include <sstream>

template <typename T>
class Queue {
public:
    void push(const T &value) {
        auto newLast = std::make_shared<Node>(value, nullptr);
        if (m_last != nullptr)
            m_last->next = newLast;
        m_last = newLast;
        if (m_first == nullptr)
            m_first = m_last;
    }
    std::optional<T> first() const {
        if (m_first == nullptr)
            return {};
        return m_first->value;
    }
    void pop() {
        if (m_first == nullptr)
            return;
        m_first = m_first->next;
        if (m_first == nullptr)
            m_last = nullptr;
    }

    bool isEmpty() const {
        return m_first == nullptr;
    }

private:
    struct Node {
        T value;
        std::shared_ptr<Node> next;

        Node (T value, decltype(next) next)
            : value(value), next(next)
        {}
```

```
    };
    std::shared_ptr<Node> m_first, m_last;
};

template<typename T>
class BinTree {
public:
    BinTree(const T &value)
        : m_value(value)
    {}

    T rootValue() const {
        return m_value;
    }
    void setRootValue(const T &value) {
        m_value = value;
    }

    std::shared_ptr<BinTree<T>> &left() {
        return m_left;
    }
    std::shared_ptr<BinTree<T>> &right() {
        return m_right;
    }

private:
    T m_value;
    std::shared_ptr<BinTree<T>> m_left, m_right;
};

enum class BinTreeLoadingStatus {
    SUCCESS,
    EXPECTED_OPENING_BRACKET,
    EXPECTED_CLOSING_BRACKET,
    UNEXPECTED_END
};
BinTreeLoadingStatus loadBinTree(std::istringstream &treeStream,
std::shared_ptr<BinTree<std::string>> &tree) {
    auto &stream = treeStream;

    auto oldExceptionState = stream.exceptions();
    stream.exceptions(std::ios_base::failbit |
std::ios_base::badbit | std::ios_base::eofbit);
```

```
    auto statusToReturn = [&stream, oldExceptionState]
(BinTreeLoadingStatus status) {
        stream.exceptions(oldExceptionState);
        return BinTreeLoadingStatus{status};
    };

    try {
        char c = 0;
        auto skipSpaces = [&c, &stream] {
            do {
                stream >> c;
            } while (isspace(c));
            stream.unget();
        };

        skipSpaces();
        stream >> c;
        if (c == '#')
            return statusToReturn(BinTreeLoadingStatus::SUCCESS);
        if (c != '(')
            return
statusToReturn(BinTreeLoadingStatus::EXPECTED_OPENING_BRACKET);

        auto readValue = [&c, &stream, skipSpaces] {
            skipSpaces();
            stream >> c;
            std::string value;
            while (!isspace(c) && c != '(' && c != ')' && c !=
'#') {
                value += c;
                stream >> c;
            }
            stream.unget();
            return value;
        };
        auto value = readValue();
        if (tree == nullptr)
            tree = std::make_shared<BinTree<std::string>>(value);
        else
            tree->setRootValue(value);

        auto tryLoadSubTree = [&stream]
(std::shared_ptr<BinTree<std::string>> &subtree) {
            auto pos = stream.tellg();
```

```
            auto status = loadBinTree(stream, subtree);
            if (status != BinTreeLoadingStatus::SUCCESS)
                stream.seekg(pos);
        };
        tryLoadSubTree(tree->left());
        tryLoadSubTree(tree->right());

        skipSpaces();
        stream >> c;
        if (c != ')')
            return
statusToReturn(BinTreeLoadingStatus::EXPECTED_CLOSING_BRACKET);

        return statusToReturn(BinTreeLoadingStatus::SUCCESS);
    } catch (std::ios_base::failure &) {
        return
statusToReturn(BinTreeLoadingStatus::UNEXPECTED_END);
    }
}


template <typename T>
std::string binTreeByLayers(std::shared_ptr<BinTree<T>> tree) {
    if (tree == nullptr)
        return "";

    std::ostringstream resultStream;

    Queue<std::shared_ptr<BinTree<T>>> q;
    q.push(tree);
    while (!q.isEmpty()) {
        auto tree = q.first().value();

        resultStream << tree->rootValue() << ' ';

        if (tree->left() != nullptr)
            q.push(tree->left());
        if (tree->right() != nullptr)
            q.push(tree->right());

        q.pop();
    }

    auto resultStr = resultStream.str();
```

```
    resultStr.pop_back();
    return resultStr;
}

int main() {
    std::shared_ptr<BinTree<std::string>> tree;

    std::cout << "Enter a bin tree: ";
    std::string treeStr;
    std::getline(std::cin, treeStr);
    std::istringstream treeStream(treeStr);

    auto loadingStatus = loadBinTree(treeStream, tree);

    switch (loadingStatus) {
    case BinTreeLoadingStatus::SUCCESS:
        if (static_cast<size_t>(treeStream.tellg()) !=
treeStr.length())
            std::cerr << "Warning: extra characters after the tree
expression."
                      << std::endl;
        std::cout << "The tree by layers: " <<
binTreeByLayers(tree) << std::endl;
        break;
    case BinTreeLoadingStatus::EXPECTED_OPENING_BRACKET:
        std::cerr << "Error: expected the opening bracket at the
position "
                  << treeStream.tellg() << '.' << std::endl;
        break;
    case BinTreeLoadingStatus::EXPECTED_CLOSING_BRACKET:
        std::cerr << "Error: expected the closing bracket at the
position "
                  << treeStream.tellg() << '.' << std::endl;
        break;
    case BinTreeLoadingStatus::UNEXPECTED_END:
        std::cerr << "Error: unexpected the end of the tree
expression."
                  << std::endl;
        break;
    }

    if (loadingStatus == BinTreeLoadingStatus::SUCCESS)
        return EXIT_SUCCESS;
    else
```

```
        return EXIT_FAILURE;
}
```